Pattern Recognition Systems – Lab 1

Least Mean Squares

1. Objectives

This laboratory work introduces the OpenCV-based framework used throughout the course. In this assignment a line is fitted to a set of points using the Least Mean Squares method (linear regression). Both the iterative solution (gradient descent) and the closed form are presented.

2. Theoretical Background

You are given a set of data points of the form (x_i, y_i) where $i = \{1,2,\dots,n\}$. Your task is to find the line equation that best fits the data. We will tackle this with linear regression. The set of points is considered the training set and your task is to learn a model that best fits the data.

Model 1

When trying to fit a model to data the first step is to establish the form of the model. Linear regression adopts a model that is linear in terms of the parameters (including a constant term). For the first part let us adopt a simple model that expresses y in terms of x:

$$f(x) = \theta_0 + \theta_1 x$$

This is the usual way this problem is solved. However, this representation cannot treat vertical lines. Nonetheless, it provides a good introduction to the method. A vector can be formed that contains all the parameters of the model $\boldsymbol{\theta} = [\theta_0, \theta_1]^T$ (the intercept term and the linear coefficient for x).

The least squares approach for determining the parameters states that the best fit to the model will be obtained when the quadratic cost function is at its minimum:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

Why quadratic? This can be motivated by making the assumption that the error in the data follows a normal distribution. See reference. Note that, this minimizes the error only along the y-axis and not the distances of the points from the line. In order to minimize the cost function we take its partial derivatives with respect to each parameter.

$$\frac{\partial}{\partial\theta_0}J(\theta) = \sum_{i=1}^{n}(f(x_i) - y_i)$$

$$\frac{\partial}{\partial\theta_1}J(\theta) = \sum_{i=1}^{n}(f(x_i) - y_i)\,x_i$$

The cost function attains its minimum when the gradient becomes zero. One general approach to find the minimum is to use **gradient descent**. Since the gradient

shows the direction in which the function increases the most, if we take steps into the opposite direction we decrease the value of the function. By controlling the size of the step we can arrive at a local minimum of the function. Since the objective function in this case is quadratic, the function has a single minimum and so gradient descent will find it.

To apply gradient descent start from an initial non-zero guess $\boldsymbol{\theta}$ chosen randomly. Find the gradient in that point:

$$\nabla J(\boldsymbol{\theta}) = \left[\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_0}, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1}\right]^T$$

Then apply the following update rule until convergence:

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta} - \alpha \nabla J(\boldsymbol{\theta}),$$

where α is the learning rate and it is chosen appropriately to ensure the cost function decreases at each iteration. When the change between the parameter values is small enough, the algorithm stops.

The gradient descent approach is appropriate when the roots of the gradient are hard to find. But in this case (line fitting) an explicit solution can be found. By setting the gradient components equal to 0 we obtain the following system:

$$\begin{cases} \theta_0 n + \theta_1 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \\ \theta_0 \sum_{i=1}^{n} x_i + \theta_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i \end{cases}$$

which is a linear system with two equations and two unknowns and can be solved directly to obtain the values for $\boldsymbol{\theta}$:

$$\begin{cases} \theta_1 = \dfrac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \\ \theta_0 = \dfrac{1}{n}\left(\sum_{i=1}^{n} y_i - \theta_1 \sum_{i=1}^{n} x_i\right) \end{cases}$$

Normal Equation - A closed form solution in vector form

In general the minimization problem for this model can be written in matrix form:

$$\|A\boldsymbol{\theta} - \boldsymbol{b}\|^2 = (A\boldsymbol{\theta} - \boldsymbol{b})^T (A\boldsymbol{\theta} - \boldsymbol{b})$$

This case arises for model 1, where A is an *nx2* matrix with each row containing the value 1 followed by the values for x_i and b is an *nx1* column vector containing the values for y. then the closed form solution is given directly by:

$$\theta_{opt} = (A^T A)^{-1} A^T \boldsymbol{b}$$

For more details and derivation consult [1].

Model 2

In order to address the issue of vertical lines we introduce another model that is capable of dealing with every possible case. Consider the following parametrization of a line in 2D:

$$xcos(\beta) + ysin(\beta) = \rho$$

This describes a line with unit normal vector $[cos(\beta), sin(\beta)]$ which is at a distance of ρ from the origin.

The cost function we wish to minimize in this case is the sum of squared distances of each point from our current line. This is given by:

$$J(\beta, \rho) = \frac{1}{2}\sum_{i=1}^{n}(x_i cos(\beta) + y_i sin(\beta) - \rho)^2$$

Note, that this is the actual error term we want to minimize and that in the previous section we have considered only the error along the y-axis, which is incorrect.

The components of the gradient need to be evaluated to perform gradient descent:

$$\frac{\partial J}{\partial \beta} = \sum_{i=1}^{n}(x_i cos(\beta) + y_i sin(\beta) - \rho)\big(-x_i sin(\beta) + y_i cos(\beta)\big)$$

$$\frac{\partial J}{\partial \rho} = -\sum_{i=1}^{n}(x_i cos(\beta) + y_i sin(\beta) - \rho)$$

A closed form solution can also be obtained, although not as easily as in the previous case. The solution is given as:

$$\beta = -\frac{1}{2}atan2\left(2\sum_{i=1}^{n}x_i y_i - \frac{2}{n}\sum_{i=1}^{n}x_i \sum_{i=1}^{n}y_i, \sum_{i=1}^{n}(y_i^2 - x_i^2) + \frac{1}{n}\left(\sum_{i=1}^{n}x_i\right)^2 - \frac{1}{n}\left(\sum_{i=1}^{n}y_i\right)^2\right)$$

$$\rho = \frac{1}{n}\left(\cos(\beta)\sum_{i=1}^{n}x_i + sin(\beta)\sum_{i=1}^{n}y_i\right)$$

Model 3

There is a third possibility for the form of the model. If we adopt a parametrization with 3 free parameters for a line:

$$ax + by + c = 0$$

The cost function can be defined as:

$$J(a, b, c) = \frac{1}{2}\sum_{i=1}^{n}(ax_i + by_i + c)^2$$

which can be written in matrix form as the squared norm of a vector:

$$J(a, b, c) = (A\boldsymbol{\theta})^T A\boldsymbol{\theta}$$

where *A* is a matrix with *nx3* elements, each row containing $(x_i, y_i, 1)$ and $\boldsymbol{\theta} = [a, b, c]^T$ is the parameter vector (column vector with 3 elements).

We need to minimize this norm to obtain the parameter values. Working with this model which has 3 parameters has two important consequences: all possible lines can be modeled and we will have a family of values which give the same line. To solve the second issue we will seek the parameter vector with unit norm. Finding the null-space of

a matrix A with unit norm is a classical problem and it is solved with Singular Value Decomposition. We have:

$$A = USV$$

where U and V are orthogonal matrices and S contains values only on the main diagonal (called singular values). From here the optimal value for the parameter vector will correspond to the last column of the matrix V (which is the eigenvector of A^TA corresponding to the smallest eigenvalue). The interested reader can consult [2] for a demonstration and further details.

3. Example results

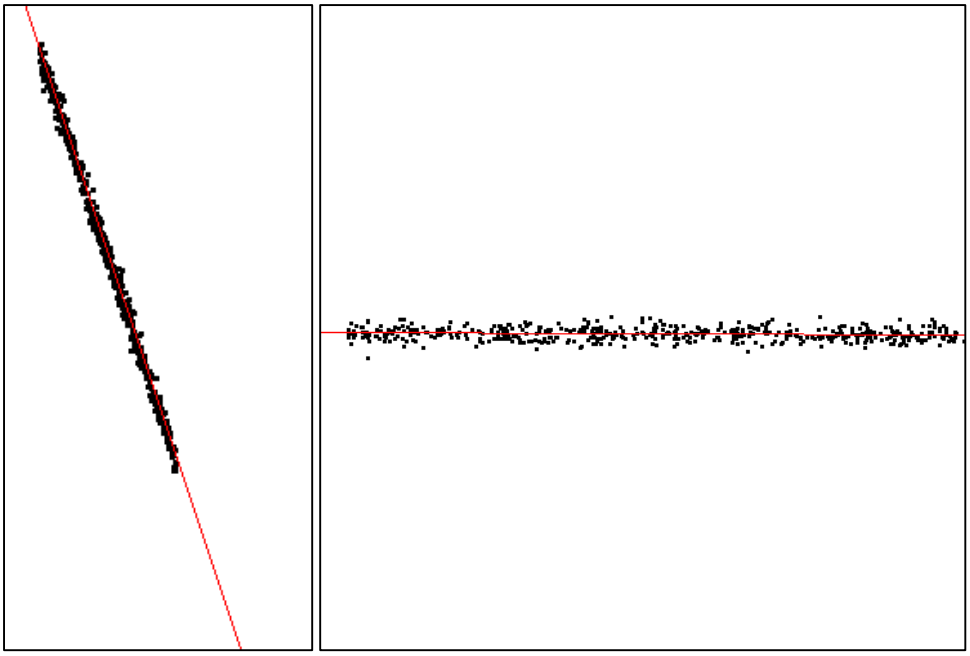

Figure 1 – Example results using model 2 on data from points1 and points2

4. Practical Background

Reading from a text file:

```
FILE* f = fopen("filename.txt","r");
float x,y;
fscanf(f,"%f%f", &x,&y);
fclose(f);
```

Creating a color image:

```
Mat img(height, width, CV_8UC3); //8bit unsigned 3 channel
```

Accessing the pixel at position row *i* and column *j*:

```
Vec3b pixel = img.at<Vec3b>(i,j); //byte vector with 3
elements
```

Modifying the pixel at row *i* and column *j*:

```
img.at<Vec3b>(i,j)[0] = 255; //blue
img.at<Vec3b>(i,j)[1] = 255; //green
img.at<Vec3b>(i,j)[2] = 255; //red
```

Draw a line between two points:

```
line(img, Point(x1, y1), Point(x2, y2), Scalar(B,G,R));
```

Viewing the image:

```
imshow("title", img);
waitKey();
```

5. Practical Work

1. Read the input data from the given file. The first line contains the number of points. Each line afterwards contains an (x,y) pair.
2. Plot the points on a white 500x500 background image. For better visibility draw circles, crosses or squares centered at the points. Be careful to consider how the coordinate system in the image is defined. Some points may have negative coordinates, either don't plot them or shift the whole graph. The method is not affected by points having negative coordinates.
3. Optionally, use *model 1* and gradient descent to fit a line to the points. Visualize the line at each k-th step. Output and visualize the value of the cost function at each step. Choose the learning rate so that the cost function is decreasing.
4. Use *model 1* and the closed form to calculate the parameters. Visualize both the final line from the previous step and this one and compare the parameter values.
5. Optionally, use *model 2* and gradient descent to fit a line to the points. Visualize the line at each k-th step. Output and visualize the value of the cost function at each step. Choose the learning rate so that the cost function is decreasing.
6. Use *model 2* and the closed form to calculate the parameters. Compare the results with the previous step.
7. Optionally, draw the errors as perpendiculars segments from the points to the line.
8. Optionally, find the parameters with *model 3* and SVD.

6. References

[1] Stanford Machine Learning - course notes 1 – http://cs229.stanford.edu/notes/cs229-notes1.pdf
[2] Tomas Svoboda - Least-squares solution of Homogeneous Equations - http://cmp.felk.cvut.cz/cmp/courses/XE33PVR/WS20072008/Lectures/Supporting/constrained_lsq.pdf